**WEEK-2**

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**CODE:**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

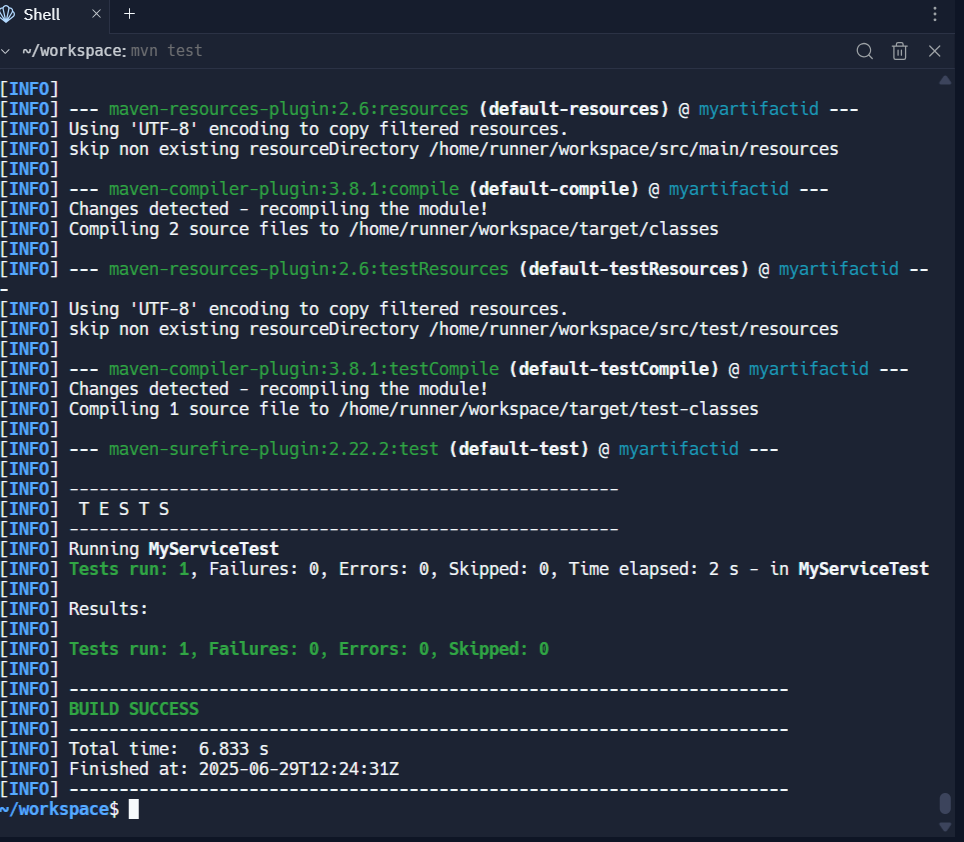
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**

****

**Exercise 2: Verifying Interactions**

**CODE:**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

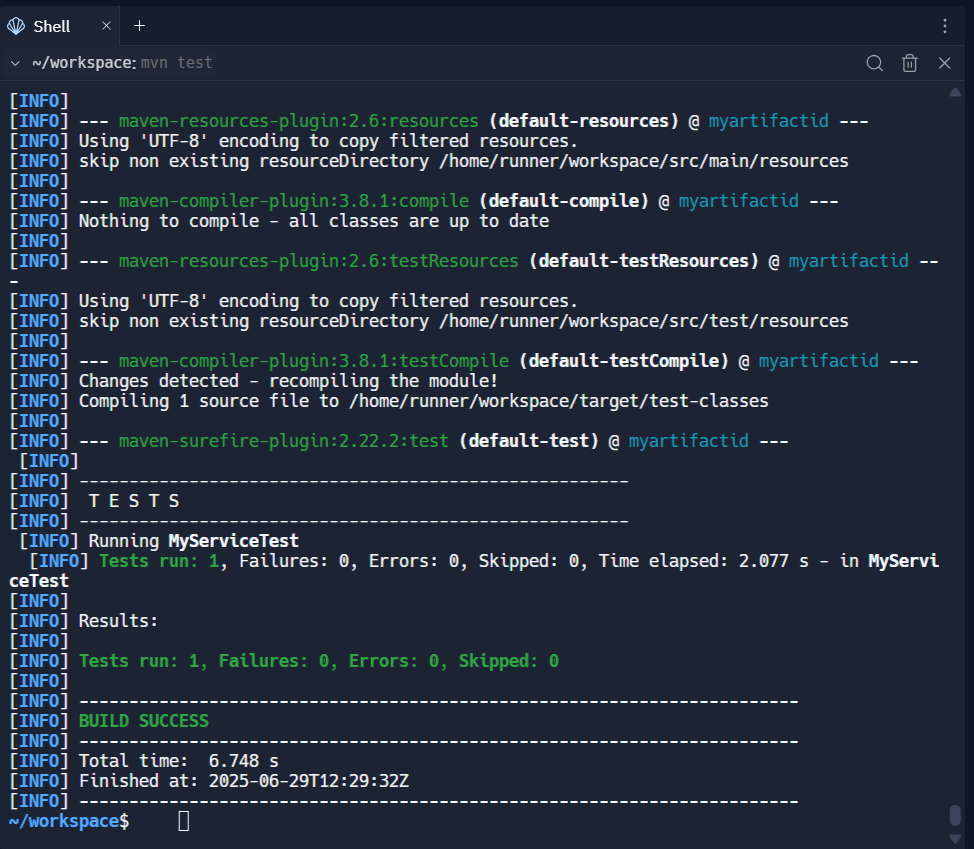
service.fetchData();

verify(mockApi).getData();

}

}

**OUTPUT:**

****

**Exercise 3: Argument Matching**

**CODE:**

**ExternalApi.java:**

public interface ExternalApi {

void sendData(String data);

}

**MyService.java:**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void processData(String input) {

String processed = input.trim().toUpperCase();

api.sendData(processed);

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.eq;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testArgumentMatching() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

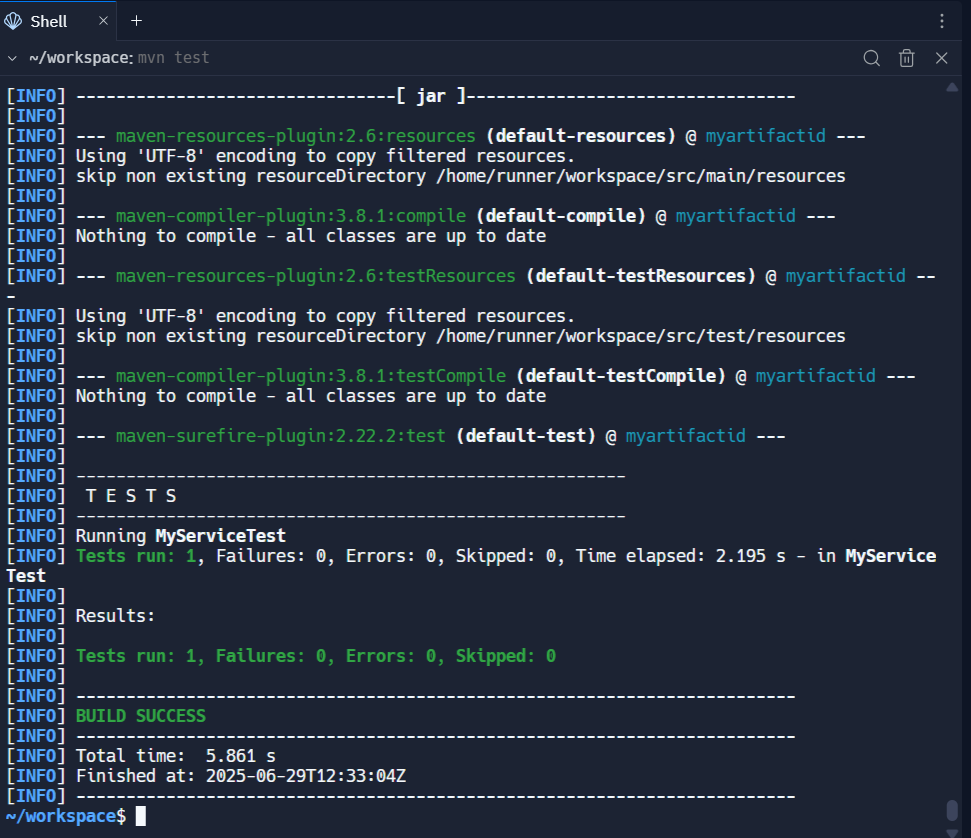
service.processData(" hello ");

verify(mockApi).sendData(eq("HELLO"));

}

}

**OUTPUT:**

****

**Exercise 4: Handling Void Methods**

**CODE:**

**NotificationService.java:**

public interface NotificationService {

void sendNotification(String message);

}

**OrderProcessor.java:**

public class OrderProcessor {

private final NotificationService notificationService;

public OrderProcessor(NotificationService notificationService) {

this.notificationService = notificationService;

}

public void processOrder(String orderId) {

// Imagine processing logic here

notificationService.sendNotification("Order " + orderId + " processed.");

}

}

**OrderProcessorTest.java:**

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

public class OrderProcessorTest {

@Test

public void testVoidMethodInteraction() {

NotificationService mockService = Mockito.mock(NotificationService.class); doNothing().when(mockService).sendNotification(anyString());

OrderProcessor processor = new OrderProcessor(mockService);

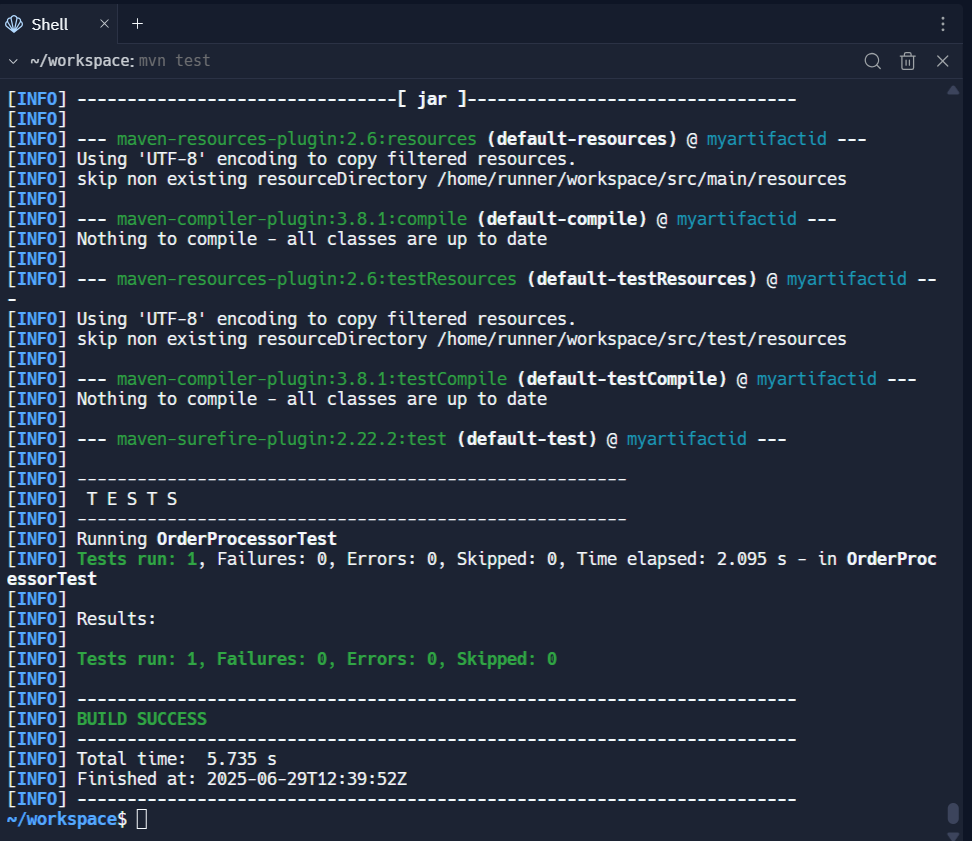
processor.processOrder("123");

verify(mockService).sendNotification("Order 123 processed.");

}

}

**OUTPUT:**

****

**Exercise 5: Mocking and Stubbing with Multiple Returns.**

**CODE:**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testMultipleReturns() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("First Call", "Second Call", "Third Call");

MyService service = new MyService(mockApi);

assertEquals("First Call", service.fetchData());

assertEquals("Second Call", service.fetchData());

assertEquals("Third Call", service.fetchData());

}

}

**OUTPUT:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Exercise 6: Verifying Interaction Order**

**CODE:**

**PaymentService.java:**

public interface PaymentService {

void initialize();

void processPayment();

void sendReceipt();

}

**OrderService.java:**

public class OrderService {

private final PaymentService paymentService;

public OrderService(PaymentService paymentService) {

this.paymentService = paymentService;

}

public void completeOrder() {

paymentService.initialize();

paymentService.processPayment();

paymentService.sendReceipt();

}

}

**OrderServiceTest.java:**

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import static org.mockito.Mockito.\*;

public class OrderServiceTest {

@Test

public void testInteractionOrder() {

PaymentService mockPaymentService = mock(PaymentService.class);

OrderService orderService = new OrderService(mockPaymentService);

orderService.completeOrder();

InOrder inOrder = inOrder(mockPaymentService);

inOrder.verify(mockPaymentService).initialize();

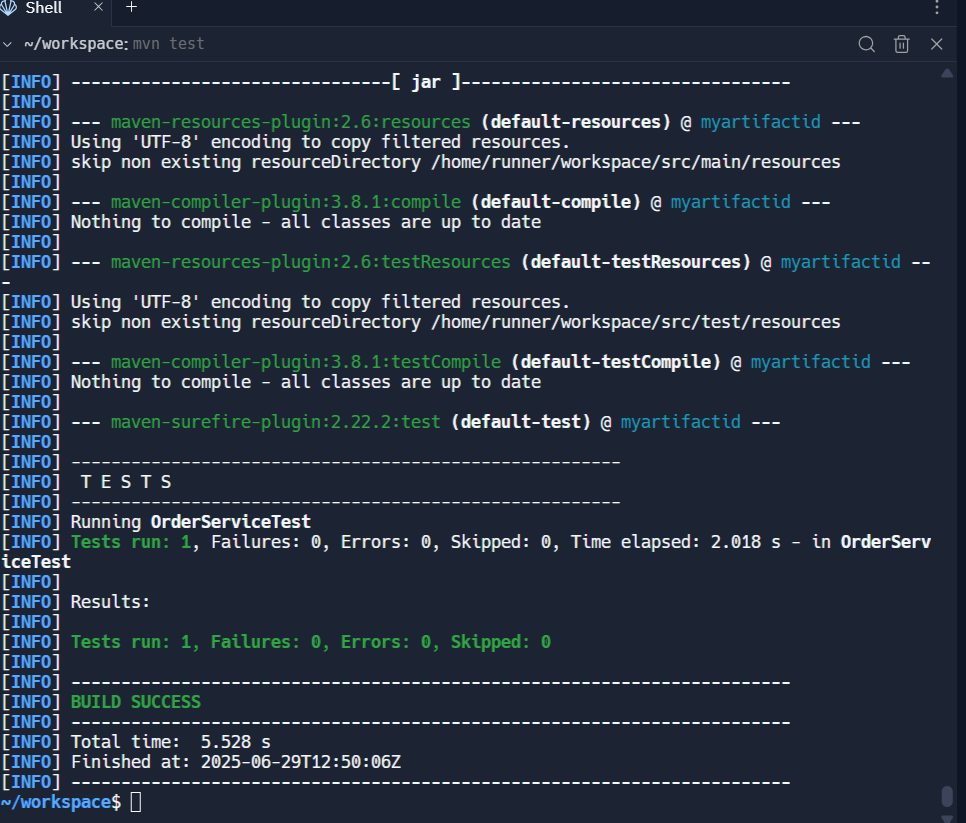
inOrder.verify(mockPaymentService).processPayment();

inOrder.verify(mockPaymentService).sendReceipt();

}

}

**OUTPUT:**

****

**Exercise 7: Handling Void Methods with Exceptions**

**CODE:**

**Logger.java:**

public interface Logger {

void log(String message) throws RuntimeException;

}

**ErrorHandler.java:**

public class ErrorHandler {

private Logger logger;

public ErrorHandler(Logger logger) {

this.logger = logger;

}

public void handleError(String error) {

try {

logger.log(error);

} catch (RuntimeException e) {

System.out.println("Logging failed: " + e.getMessage());

}

}

}

**ErrorHandlerTest.java:**

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

public class ErrorHandlerTest {

@Test

public void testVoidMethodThrowsException() {

Logger mockLogger = mock(Logger.class);

doThrow(new RuntimeException("Log failed")).when(mockLogger).log("Critical Error");

ErrorHandler handler = new ErrorHandler(mockLogger);

handler.handleError("Critical Error");

verify(mockLogger).log("Critical Error");

}

}

**OUTPUT:**

**A screenshot of a computer program

AI-generated content may be incorrect.**